Winrm Active Directory Diagnostics using HTTPS

winrm_https_diag.py

```
#!/usr/bin/env python3
"""
WinRM/HTTPS diagnostic — run this when winrm_ad_enum.py returns nothing.

Tests each layer separately and prints raw output at every step:
    1. TCP reachability on 5986
    2. TLS handshake + cert details
    3. WinRM session + trivial command ($env:COMPUTERNAME)
    4. ActiveDirectory module availability on the remote host
    5. Raw user count (single integer, easiest thing to debug)
    6. Single-user JSON sample (so we can see what PowerShell actually returns)

Usage:
    python winrm_https_diag.py --host dc01.corp.local --user 'CORP\\admin'
"""
import argparse
import getpass
import socket
import ssl
import sys
from contextlib import closing

import winrm

GREEN = "\033[92m"; RED = "\033[91m"; YEL = "\033[93m"; BLUE = "\033[94m"; END = "\033[0m"

def step(n, title):
    print(f"\n{BLUE}── STEP {n}: {title} ──{END}")

def ok(msg):   print(f"{GREEN}[OK]{END}   {msg}")
def bad(msg):  print(f"{RED}[FAIL]{END} {msg}")
```

```
def warn(msg): print(f"{YEL}[WARN]{END} {msg}")



def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--host", required=True)
    ap.add_argument("--user", required=True)
    ap.add_argument("--password")
    ap.add_argument("--port", type=int, default=5986)
    ap.add_argument("--transport", default="ntlm",
                    choices=["ntlm", "kerberos", "ssl", "credssp"])
    args = ap.parse_args()
    pw = args.password or getpass.getpass(f"Password for {args.user}: ")

    # ---------- Step 1: TCP ----------
    step(1, f"TCP connectivity to {args.host}:{args.port}")
    try:
        with closing(socket.create_connection((args.host, args.port), timeout=5)):
            ok(f"TCP {args.port} is open")
    except Exception as e:
        bad(f"TCP failed: {type(e).__name__}: {e}")
        return 1

    # ---------- Step 2: TLS ----------
    step(2, "TLS handshake")
    ctx = ssl.create_default_context()
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE
    try:
        with socket.create_connection((args.host, args.port), timeout=5) as s:
            with ctx.wrap_socket(s, server_hostname=args.host) as ss:
                cipher = ss.cipher()
                cert = ss.getpeercert(binary_form=True)
                ok(f"TLS handshake succeeded. Cipher: {cipher}")
                ok(f"Server presented a cert ({len(cert)} bytes DER)")
    except Exception as e:
        bad(f"TLS handshake failed: {type(e).__name__}: {e}")
        return 1
```

```
    # ---------- Step 3: WinRM session + trivial command ----------
    step(3, f"WinRM session ({args.transport})")
    endpoint = f"https://{args.host}:{args.port}/wsman"
    print(f"      Endpoint: {endpoint}")
    try:
        session = winrm.Session(
            endpoint,
            auth=(args.user, pw),
            transport=args.transport,
            server_cert_validation="ignore",
        )
    except Exception as e:
        bad(f"Could not build session: {type(e).__name__}: {e}")
        return 1

    try:
        r = session.run_ps("$env:COMPUTERNAME")
        out = r.std_out.decode(errors="replace").strip()
        err = r.std_err.decode(errors="replace").strip()
        print(f"      exit_code = {r.status_code}")
        print(f"      stdout    = {out!r}")
        print(f"      stderr    = {err!r}")
        if r.status_code == 0 and out:
            ok(f"WinRM works. Remote computer = {out}")
        else:
            bad("WinRM responded but command failed. Check stderr above.")
            return 1
    except Exception as e:
        bad(f"WinRM command failed: {type(e).__name__}: {e}")
        return 1

    # ---------- Step 4: AD module ----------
    step(4, "ActiveDirectory PowerShell module")
    probe = r"""
$ErrorActionPreference = 'Stop'
try {
    Import-Module ActiveDirectory -ErrorAction Stop
    Write-Output "MODULE_OK"
    Write-Output ("AD module version: " + (Get-Module ActiveDirectory).Version)
```

```
} catch {
    Write-Output "MODULE_MISSING: $($_.Exception.Message)"
    exit 2
}
"""
    r = session.run_ps(probe)
    out = r.std_out.decode(errors="replace").strip()
    err = r.std_err.decode(errors="replace").strip()
    print(f"      exit_code = {r.status_code}")
    print(f"      stdout    = {out!r}")
    print(f"      stderr    = {err!r}")
    if "MODULE_OK" not in out:
        bad("ActiveDirectory module is NOT available on the remote host.")
        print("      Fix: connect to a Domain Controller, or install RSAT-AD-PowerShell")
        print("      on the target box: Install-WindowsFeature RSAT-AD-PowerShell")
        return 1
    ok("AD module loaded")

    # ---------- Step 5: Raw counts ----------
    step(5, "Raw object counts (sanity check)")
    count_script = r"""
$ErrorActionPreference = 'Stop'
Import-Module ActiveDirectory
$u = @(Get-ADUser -Filter *).Count
$c = @(Get-ADComputer -Filter *).Count
Write-Output "USERS=$u"
Write-Output "COMPUTERS=$c"
"""
    r = session.run_ps(count_script)
    out = r.std_out.decode(errors="replace").strip()
    err = r.std_err.decode(errors="replace").strip()
    print(f"      exit_code = {r.status_code}")
    print(f"      stdout    = {out!r}")
    print(f"      stderr    = {err!r}")
    if r.status_code != 0:
        bad("Count query failed — see stderr above.")
        return 1
    if "USERS=0" in out and "COMPUTERS=0" in out:
        warn("Both counts are 0. The account may lack read permission on AD,")
```

```
        warn("or the directory is genuinely empty (unlikely on a real DC).")
    else:
        ok("Counts retrieved successfully")

    # ---------- Step 6: Single-user JSON sample ----------
    step(6, "Single-user JSON sample (this is what the main script parses)")
    sample = r"""
$ErrorActionPreference = 'Stop'
Import-Module ActiveDirectory
$one = Get-ADUser -Filter * -ResultSetSize 1 -Properties DisplayName, Enabled, LastLogonDate |
    Select-Object SamAccountName, DisplayName, Enabled,
        @{N='LastLogonDate';E={if($_.LastLogonDate){$_.LastLogonDate.ToString('o')}}}
,@($one) | ConvertTo-Json -Depth 4 -Compress
"""
    r = session.run_ps(sample)
    out = r.std_out.decode(errors="replace")
    err = r.std_err.decode(errors="replace").strip()
    print(f"      exit_code = {r.status_code}")
    print(f"      stdout (first 500 chars):")
    print(f"      {out[:500]!r}")
    if err:
        print(f"      stderr    = {err!r}")
    if r.status_code == 0 and out.strip():
        ok("Sample JSON received — the main script's parsing should work.")
    else:
        bad("No JSON output. The ConvertTo-Json step is failing.")
        return 1

    print(f"\n{GREEN}All six steps passed. The HTTPS path is fully working.{END}")
    print("If the main enumeration script still returns nothing, please share")
    print("its exact output — the issue is elsewhere.")
    return 0


if __name__ == "__main__":
    sys.exit(main())
```

Step 1 — Tell me what you actually see

Could you tap which output pattern matches what you're seeing?

While you pick, here's a **quick diagnostic script** that isolates each layer so we can pinpoint exactly where things break. Run this before changing anything in the main script:

Created a file, read a file
Created a file, read a file

Run the diagnostic

bash

```
python winrm_https_diag.py --host YOUR_DC --user 'DOMAIN\admin'
```

It runs **six layered checks** and prints raw `exit_code`, `stdout`, and `stderr` at every step. The first one that fails tells us exactly where the problem is.

Most likely causes, ranked by what I see most often

While you run the diagnostic, here are the usual suspects for "HTTPS returns nothing":

1. HTTPS listener not configured on the server (most common) The Windows Server has WinRM enabled but only on port 5985. Verify on the server:

powershell

```
winrm enumerate winrm/config/Listener
```

You should see two entries — one with `Transport = HTTP` and one with `Transport = HTTPS`. If only HTTP shows up, follow Part 2 of my earlier guide to create the HTTPS listener.

2. Firewall blocking 5986 specifically Default Windows firewall rules cover 5985 but **not 5986**. Check on the server:

powershell

```
Get-NetFirewallRule -DisplayName "*WinRM*HTTPS*"
```

If nothing comes back, add it:

powershell

```
New-NetFirewallRule -DisplayName "WinRM HTTPS-In" -Direction Inbound `
    -LocalPort 5986 -Protocol TCP -Action Allow
```

3. PowerShell `ConvertTo-Json` silently choking On some PowerShell versions, `ConvertTo-Json -Depth 4` truncates objects with deeper nesting and emits **nothing** instead of an error. The diagnostic's Step 6 will catch this — if `stdout` is empty there, that's the issue.

4. Credentials work for auth but the account can't read AD A standard user account can authenticate to WinRM but may not have permission to read user/computer attributes. Step 5 of the diagnostic prints the raw counts — if both are 0 against a real DC, this is it.

5. The user is logged in as a local account, not a domain account `--user admin` is treated as a local account. For AD queries you need a domain account: `--user 'CORP\admin'` or `--user admin@corp.local`.

6. Network proxy or HTTPS inspection appliance A corporate proxy doing TLS inspection will break NTLM message integrity. The diagnostic's Step 3 will fail with a cryptic error if this is happening — usually `HTTPSConnectionPool` or `BadStatusLine`.

Once you've either picked an option above or run the diagnostic, paste me what you see and I'll narrow it down further.

Revision #1
Created 19 May 2026 16:04:24 by Admin
Updated 19 May 2026 16:05:35 by Admin